Evolution of Metaparameters for Efficient Real Time Learning

Genci Capi[1], Masao Yokota[1] and Kenji Doya[2]

[1]Faculty of Information Engineering
Fukuoka Institute of Technology
3-30-1 Wajiro-Higashi, Higashi-ku, Fukuoka, 811-0195, Japan

[2] CREST, Japan Science and Technology Agency (JST)
ATR, Computational Neuroscience Laboratories
"Keihanna Science City", Kyoto, 619-0288, Japan

capi@fit.ac.jp

Abstract
In this paper, we propose a new method based on evolutionary computation for setting the metaparameters of reinforcement learning in order to match the demands of the task and reduce the learning time. The basic idea is to encode the metaparameters of the reinforcement learning algorithm as the agent's genes, and to take the metaparameters of best-performing agents in the next generation. We considered a complex task where the Cyber Rodent robot has to survive and increase its energy level. The results show that appropriate settings of metaparameters found by evolution have a great effect on the learning time and are strongly dependent on each other.

KEY WORDS
Actor-critic Reinforcement Learning, Genetic Algorithm, metaparameters.

1. Introduction

Reinforcement learning (RL) ([1], [2],[3], [4]) provides a sound framework for autonomous agents to acquire adaptive behaviors based on reward feedback. The theory of RL has been successfully applied to a variety of dynamic optimization problems, such as game programs ([5]), and resource allocation.

In RL, the learning capabilities are strongly dependant on a number of parameters, such as learning rate, the degree of exploration, and the time scale of evaluation. The appropriate settings of metaparameters depend on the environmental dynamics, the goal of the task, and the time allowed for learning. The permissible ranges of such metaparameters are dependant on particular tasks and environments, making it necessary for a human expert to tune them usually by trial and error. But tuning multiple metaparameters is quite difficult due to their mutual dependency, e.g., if one changes the noise size, one should also change the learning rate. In addition hand tuning of metaparameters is in a marked contrast with learning in animals, which can adjust themselves to unpredicted environments without any help from a supervisor.

The specific questions we ask in this study are: 1) whether GA can successfully find appropriate metaparameters subject to mutual dependency and 2) how the metaparameters and initial weight connections effect the learning time. In our method, the basic idea is to encode the metaparameters of the RL algorithm as the agent's genes, and to take the metaparameters of best-performing agents in the next generation.

In order to answer these questions, we considered a surviving behavior for the Cyber Rodent (CR) robot ([11]), which is a two wheeled robot with a wide-angle camera. The robot must recharge itself by capturing active battery packs distributed on the environment. In order to see the effect of metaparameters and initial weight connections on learning time we considered learning with: 1) arbitrarily selected metaparameters and random initial weight connections; 2) evolved initial weight connections and arbitrarily selected metaparameters; 3) evolved metaparameters and random initial weight connections; 4) evolved metaparameters and initial weight connections.

Results show that appropriate settings of meta-parameters can be found by evolution. The learning time is significantly reduced when the agent learned using the optimized metaparameters and the initial weight connections.

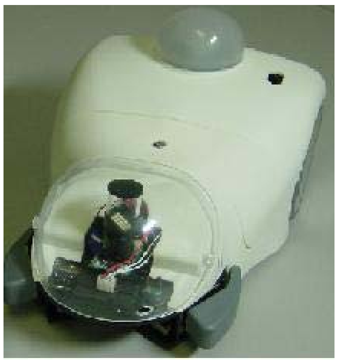

Fig. 1. Cyber Rodent robot.

2. Cyber Rodent Robot

The CR robot is a two-wheel-driven mobile robot as shown in Fig. 1. The CR is 250 mm long and weights 1.7 kg. The CR is equipped with:

- Omni-directional C-MOS camera.
- IR range sensor.
- Seven IR proximity sensors.
- 3-axis acceleration sensor.
- 2-axis gyro sensor.
- Red, green and blue LED for visual signaling.
- Audio speaker and two microphones for acoustic communication.
- Infrared port to communicate with a nearby agent.
- Wireless LAN card and USB port to communicate with the host computer.

3. Task and Environment

In the second environment, the CR robot has to survive and increase its energy level. The environment has 8 battery packs, as shown in Fig. 2. The positions of battery packs are considered fixed in the environment and the CR robot is initially placed in a random position and orientation.

The agent can recharge its own battery by capturing active battery packs, which are indicated by red LED color. After the robot captures the battery pack, it can recharge its own battery for a determined period of time (charging time), then the battery pack becomes inactive and its LED color changes to green. The battery becomes active again after the reactivation time. Therefore, in this environment the following parameters can vary:

- The number of battery packs;
- The reactivation time;
- The energy received by capturing the battery pack (by changing the charging time);
- The energy consumed by the agent for 1m motion.

Based on the energy level and the distance to the nearest active battery pack, the agent can select among three actions: 1) capture the active battery pack; 2) search for a battery pack or 3) wait until a battery pack becomes active. In the simulated environment, the batteries have a long reactivation time. In addition, the energy consumed for 1m motion is low. Therefore, the best policy is to capture any visible battery pack (the nearest when there are more than one). When there is no visible active battery pack, the agent must search in the environment.

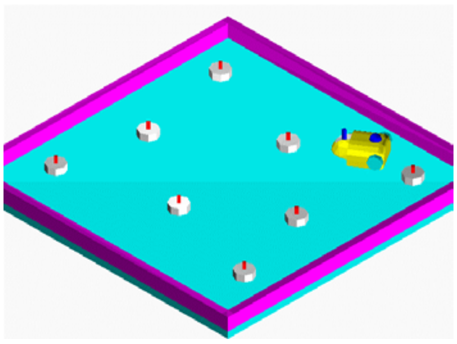

Fig. 2. Environment.

4. Actor-Critic RL

We applied an actor-critic RL. The agent can selects among three actions: 1) Capture the active battery pack; 2) Search for an active battery pack; 3) Wait for a determined period of time. The wait behavior is interrupted if a battery becomes active or after a pre-determined period of time. Both networks receive as input a constant bias input, the CR battery level and the distance to the nearest active battery pack (both normalized between 0 and 1).

4.1 The Critic

The critic has a single output cell, whose firing rate is calculated as follows:

$$O_c = \sum_{i=1}^{3} b_i x_i + \sum_{i=1}^{m} c_i y_i \qquad (1)$$

where *m* is the number of hidden neurons,

$$y_i = g\left(\sum_{j=1}^{3} a_{ij} x_j\right), \; g(s) = \frac{1}{1+e^{-s}} .$$

The TD error is calculated as follows:

$$\hat{r}[t+1] = \begin{cases} 0 \text{ if the start state} \\ \text{r}[\text{t}+1] + \gamma^{\text{k}} v[t] \text{ otherwise} \end{cases}, \quad (2)$$

using the reward

$$r_{t+1} = (En\_level_{t+1} - En\_level_t)/50.$$

TD reduces the error by changing the weights, as follows:

$$b_i[t+1] = b_i[t] + \rho_1 \hat{r}[t+1] x_i[t]$$
$$c_i[t+1] = c_i[t] + \rho_1 \hat{r}[t+1] y_i[t] \quad (3)$$
$$a_{ij}[t+1] = a_{ij}[t] + \rho_2 \hat{r}[t+1] y_i[t](1-y_i[t]) \text{sgn}(c_i[t]) x_j[t],$$

where ρ_1, ρ_2 are the learning rates.

4.2 The actor

The agent can select one of three actions and so the actor make use of three action cells, p_j, $j=1,2,3$. The captured behavior is considered pre-learned ([6]). When the search behavior is activated, the agent rotates 10 degrees clockwise. The agent does not move when the wait behaviour becomes active. The output of action neurons is calculated as follows:

$$z_i = g(\sum_{j=1}^{3} d_{ij} x_j), \quad (4)$$

$$p_i = g(\sum_{i=1}^{3} e_{ij} x_i + \sum_{i=1}^{n} f_{ij} z_i), \quad (5)$$

where $\text{g(s)} = \frac{1}{1+\exp^{-\text{s}}}$ and n is the number of hidden neurons

A winner-take-all rule prevents the actor from performing two actions at the same time. The action is selected based on the softmax method as follows:

$$P(a,s_t) = \frac{e^{p_i(a,s)/\tau}}{\sum_{i=1}^{3}(e^{p_i(a,s)/\tau})}, \quad (6)$$

where τ is the temperature of the algorithm.

Table 1. GA functions and parameters.

Function Name	Parameters
Arithmetic Crossover	2
Heuristic Crossover	[2 3]
Simple Crossover	2
Uniform Mutation	4
Non-Uniform Mutation	[4 GNmax 3]
Multi-Non-Uniform Mutation	[6 GNmax 3]
Boundary Mutation	4
Normalized Geometric Selection	0.08

The actor weights are adapted as follows:

$$e_i[t+1] = e_i[t] + \alpha_1 \hat{r}[t+1](q[t] - p[t]) x_i[t]$$
$$f_i[t+1] = f_i[t] + \alpha_1 \hat{r}[t+1](q[t] - p[t]) z_i[t] \quad (7)$$
$$d_{ij}[t+1] = d_{ij}[t] + \alpha_2 \hat{r}[t+1] z_i[t](1 - z_i[t])$$
$$\text{sgn}(f_i[t])(q[t] - p[t]) x_j[t]$$

5. Evolution of Metaparameters

In our implementation, a real-value GA was employed in conjunction with the selection, mutation and crossover operators (Table 1).

The fitness function in the surviving behavior is considered as follows:

$$Fitness = \begin{cases} \dfrac{CR_{life}}{learn\_t} & \text{if aget dies} \\ En_{level} + 1 & \text{if agent survives} \\ En_{\max} + \dfrac{learn\_t}{CR_{life}} & \text{battery fully recharged} \end{cases}$$

where CR_{life} is the CR life in seconds, *max_learning_time* is the maximum learning time, En_{level} is the level of energy if the agent survives but the battery is not fully recharged and En_{max} is the maximum level of energy. The maximum learning time is 7200 s. The value of En_{max} is 1 and En_{level} varies between 0 and 1. Based on this fitness function the agents that can not survive get a better fitness if they live longer. The agents that survive get a better fitness if the energy level at the end of maximum learning time is higher. The agents that learned to fully recharge their battery faster get the highest fitness value.

In order to see the effect of metaparameters and initial weight connections on learning time, we considered the following cases: (a) Evolution of meta-parameters, initial weight connection of actor and critic networks, and the number of hidden neurons; (b) Evolution of meta-parameters with randomly initialized weight connections; (c) Evolution of initial weight connections with arbitrary selected meta-parameters; (d) Learning with arbitrary selected meta-parameters and randomly initialized weight connections.

6. Results

In order to determine the energy level after each action, we measured the battery level of CR when the robot moves with a nearly constant velocity of 0.3m/s. The collected data are shown in Fig. 3. The graph shows that there is a nonlinear relationship between time and energy level. Therefore, we used the virtual life time to determine the energy level after each action. Except capturing the battery pack, the search and wait actions increased the virtual life time.

First, learning took place with arbitrarily selected metaparameters (Table 2) and random initial weight connections. The agent continued learning for nearly 5500s until the battery was fully recharged (Figure 4). Then, using the same metaparameters and neural structure, we evolved the initial weight connections. Initially, 100 individuals are created and the evolution terminated after 20 generations. Figure 4 shows that learning time is reduced.

When the metaparameters are evolved and initial weight connections are randomly selected, the learning time is significantly reduced (Figure 4). This result is important because it shows that metaparameters has larger effect on learning time compared to initial weight connections. The searching interval and GA results, when both metaparameters and initial weight connections are optimized by GA, are shown in Table 3. The critic and actor networks have 1 and 3 hidden neurons, respectively. The initial value of weight connections are very near with their respective values after learning. In addition, the optimized value of cooling factor is low. Therefore, the agent starts to exploit the environment and make greedy actions soon after the learning starts, recharging its own battery in a very short time.

Table 2. Values of metaparameters

Parameters	**Values**
Initial Weights	Randomly between [-0.5 0.5]
$\rho_1, \rho_2, \alpha_1, \alpha_2$	0.2; 0.1; 0.8; 0.3
τ_0	10
γ	0.9

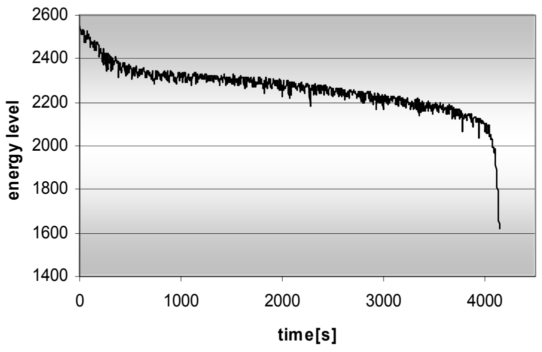

Fig. 3. Battery level during CR motion.

Table 3. Searching space and GA results.

Optimized parameters	**Searching interval**	**Results**
Initial Weights	[-1 1]	
$\rho_1, \rho_2, \alpha_1, \alpha_2$	[0 1]	0.9210; 0.3022; 0.9256; 0.6310
τ_0	[1 10]	5.3718
γ	[0 1]	0.794

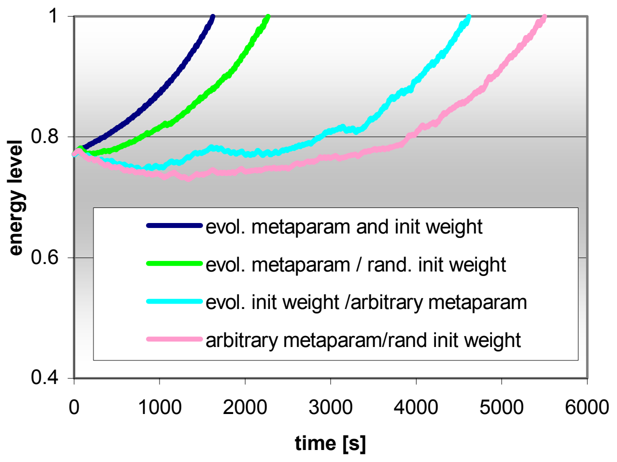

Fig. 4. Energy level of the best agent for different combinations of learning and evolution.

7. Conclusion

In this paper, we presented a method to optimize the metaparameters in RL based on evolutionary approach. Based on the simulations and experimental results we conclude:

- Evolutionary algorithms can be successfully applied to determine the optimal values of metaparameters.
- Meaparameters play important role on the agent learned policy.
- Optimized metaparameters can significantly reduce the learning time.

In the future, we are interesting to see if evolution can also be applied to shape the reward function used during the learning process.

References:

[1] Barto AG(1995), Reinforcement learning. In M. A. Arbib (Ed.), The handbook of brain theory and neural networks, (pp. 804–809). Cambridge, MA: MIT Press.
[2] Doya K (2000), Reinforcement learning in continuous time and space. Neural Computation, 12:215–245.
[3] Doya K, Kimura H, Kawato M (2001), Computational approaches to neural mechanism of learning and control. IEEE Control Systems Magazine, 21(4):42–54.
[4] Sutton RS, Barto AG (1998), Reinforcement learning. Cambridge, MA, MIT Press.
[5] Tesauro G (1994), TD-Gammon, a self teaching backgammon program, achieves master-level play. Neural Computation, 6:215–219.
[6] Capi G, Uchibe E, Doya K (2002). Selection of neural Architecture and the environment Complexity. Advanced Intelligence, Vol. 6, pp. 311-317, Editors: K. Murase and T. Asakura.